Quality of Service Scheduling in Real-Time Systems

Audrey Marchand, Maryline Chetto

Quality of Service Scheduling in Real-Time Systems

A. Marchand M. Chetto

Abstract

In this paper, we deal with dynamic scheduling components integrating new Quality of Service (QoS) functionalities into a Linux-based real-time operating system. In our approach, periodic tasks allow occasional deadline violations within given bounds specified according to the Skip-Over task model. Hence, every task has a minimal QoS guarantee which is expressed by the ratio of periodic task instances which must complete before their deadline. The work stated here provides two on-line scheduling algorithms, namely RLP and RLP/T, which enhance the existing Skip-Over algorithms. More specifically, the proposed algorithms aim at improving the actual QoS observed for periodic tasks (which is always greater or equal to the QoS guarantee). These novel scheduling techniques rely on the EDL (Earliest Deadline as Late as possible) scheduling strategy. Simulation results show the performance of RLP and RLP/T with respect to basic Skip-Over algorithms. Finally, we present the integration of these QoS scheduling services into CLEOPATRE open-source component library, a patch to Linux/RTAI.

Keywords: real-time, dynamic scheduling, quality of service, periodic tasks, component-based systems, Linux-based systems

1 Introduction

Software environments, and more precisely operating systems have still difficulties to meet the special demands of multimedia applications. In particular, multimedia applications have real-time constraints which are not handled properly by general-purpose operating systems. In order to meet the requirements imposed by multimedia applications on processor scheduling, we have to turn to the temporal stringency of real-time systems. Real-time systems are those in which the time at which the results are produced is important. The correctness of the result of a task is not only related to its logic correctness, but also to when the results occur.

Traditional classification of real-time systems stands for three classes to characterize the real-time requirement of such systems : hard, soft and firm. In hard real-time systems, all instances must be guaranteed to complete within their deadlines. In those critical control applications, missing a deadline may cause catastrophic consequences on the controlled system. For soft systems, it is acceptable to miss some of the deadlines occasionally. It is still valuable for the system to finish the task, even if it is late. In firm systems, tasks are also allowed to miss some of their deadlines, but, there is no associated value if they finish after the deadline. Typical illustrating examples of systems with firm real-time requirements are multimedia systems in which it is not necessary to meet all the task deadlines as long as the deadline violations are adequately spaced.

A prominent strategy for performing resource management for multimedia systems is QoS-driven management, in which quality requirements such as resolution and frame rate are translated into resource requirements such as computation burst frequencies and durations. This resource information is then used for admission testing and resource reservation. The motivation for this translation from application level requirements to resource requirements is to guarantee a given QoS. The complexity of both the QoS space and the resource space suggests

that perfect characterization is hard to achieve, so it would be desirable to have a scheduling policy that would adapt to changes in user QoS requirements. Such policy should strive to achieve the desired QoS, in an environment with variable resources, as well as complex and variable application demands.

In this paper, we address the problem of the dynamic scheduling of periodic tasks with firm constraints. The scope of the paper is to maximize the actual QoS of periodic tasks by maximizing the number of instances which complete before their deadline. The remainder of this paper is organized in the following manner. Next section introduces existing approaches for scheduling firm real-time systems. Then we present relevant background material about both the Skip-Over model and the EDL scheduling algorithm. More particularly, we give the definition of RTO and BWP scheduling algorithms, which are based on the Skip-Over model. The functioning and optimality of the EDL algorithm is also outlined. Further, we describe the proposed algorithms, namely RLP and RLP/T, as an enhancement of the BWP algorithm, based on the EDL scheduling mechanism. Moreover, we present a model of a real-world problem to show the practical interest of our work. The performance analysis of both RLP and RLP/T, in terms of task completions, is reported after. Then, we describe the integration of these QoS components into Linux/RTAI. Finally, in section 8, we summarize our contribution.

2 Related work

There have been some previous approaches to the specification and design of real-time systems that tolerate occasional losses of deadlines. Hamdaoui and Ramanathan in [7] introduced the concept of *(m,k)-firm* deadlines to model tasks that have to meet m deadlines every k consecutive invocations. Their algorithm uses a *distance-based priority (DBP)* scheme to increase the priority of a job in danger of missing more than m deadlines over a sliding window of k requests for service. Moreover, algorithms such as VDS [17] and DWCS [19] are provably superior to DBP in meeting (m,k) service requirements for a number of specific and non-trivial situations.

Similar to (m,k)-*firm* scheduling is the work introduced by Koren and Shasha [8] with the notion of *skip factor*. If a task has a skip factor of s, it will have one invocation skipped out of s. It is a particular case of the *(m,k)-firm* model where $m = k - 1$. They reduce the overload by skipping some task invocations, thus exploiting skips to increase the feasible periodic load. This approach gives a solution to the scheduling problem of overloaded systems, while representing a system Quality of Service requirement for real-time applications. Broadly speaking, the Skip-Over scheduling algorithms guarantee the timing correctness of the real-time application. One interesting result is that making optimal use of skips is a NP-hard problem. There are also examples of (m,k)-hard schedulers [1], but most such approaches require off-line feasibility tests, to ensure predictable service.

In [3, 4], Caccamo and Buttazzo follow this work by scheduling hybrid task sets consisting of skippable periodic and soft aperiodic tasks. They propose and analyze an algorithm, based on a variant of Earliest Deadline First (EDF) scheduling, in order to exploit skips under the Total Bandwith Server (TBS). In previous works [10, 12], we have considered the same approach but using the Earliest Deadline as Late as possible server (EDL). These results have led us to propose a raw version of the RLP algorithm (idle time schedule based on red tasks only) [11].

West and Poellabauer in [16] proposed a *windowed lost rate*, that specifies a task can tolerate x deadlines missed over a finite range or window, among consecutive y instances. In [2], Bernat *et al.* introduce a general framework for specifying tolerance of missed deadlines under the definition of *weakly hard* constraints.

Task	T_0	T_1	T_2	T_3	T_4
c_i	3	4	1	7	2
p_i	30	20	15	12	10

Table 1: A basic periodic task set

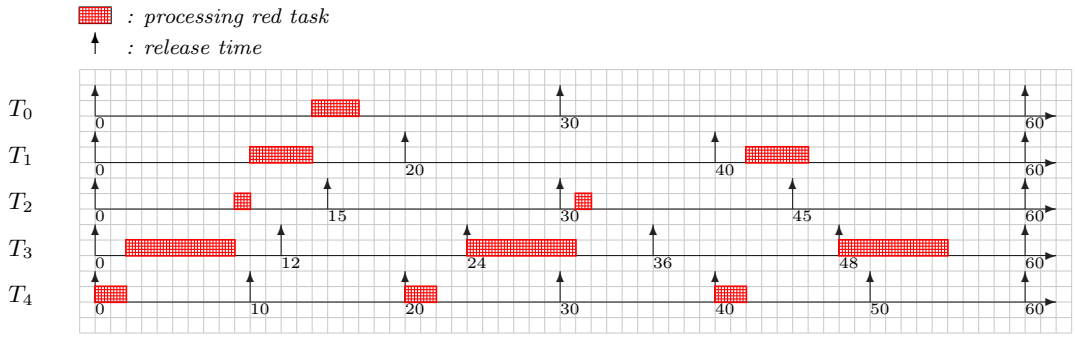

Figure 1: RTO scheduling algorithm ($s_i = 2$)

3 Theoretical Background

3.1 The Skip-Over model

We are here interested in the problem of scheduling periodic tasks which allow occasional deadline violations (*i.e.*, skippable periodic tasks), on a uniprocessor system. We assume that tasks can be preempted and that they do not have precedence constraints. A task T_i is characterized by a worst-case computation time c_i, a period p_i, a relative deadline equal to its period, and a skip parameter s_i, which gives the tolerance of this task to missing deadlines. The distance between two consecutive skips must be at least s_i periods. When s_i equals to infinity, no skips are allowed and T_i is equivalent to a hard periodic task. One can view the skip parameter as a QoS metric (the higher s_i, the better the quality of service).

A task T_i is divided into instances where each instance occurs during a single period of the task. Every instance of a task can be red or blue [8]. A red task instance must complete before its deadline; a blue task instance can be aborted at any time. However, if a blue instance completes successfully, the next task instance is still blue.

3.1.1 Red Tasks Only (RTO) algorithm

The first algorithm proposed by Koren and Shasha is the Red Tasks Only (RTO) algorithm. Red instances are scheduled as soon as possible according to Earliest Deadline First (EDF) algorithm, while blue ones are always rejected. Deadline ties are broken in favor of the task with the earliest release time. In the deeply red model where all tasks are synchronously activated and the first $s_i - 1$ instances of every task T_i are red, this algorithm is optimal. RTO is illustrated in Figure 1 using the task set $\mathcal{T} = \{T_0, T_1, T_2, T_3, T_4\}$ of five periodic tasks whose parameters are described in Table 1. Tasks have uniform skip parameter $s_i = 2$ and the total processor utilization factor $U_p = \sum \frac{c_i}{p_i}$ is equal to 1.15.

As we can see, the distance between every two skips is exactly s_i periods, thus offering only the minimal guaranteed QoS level for periodic tasks.

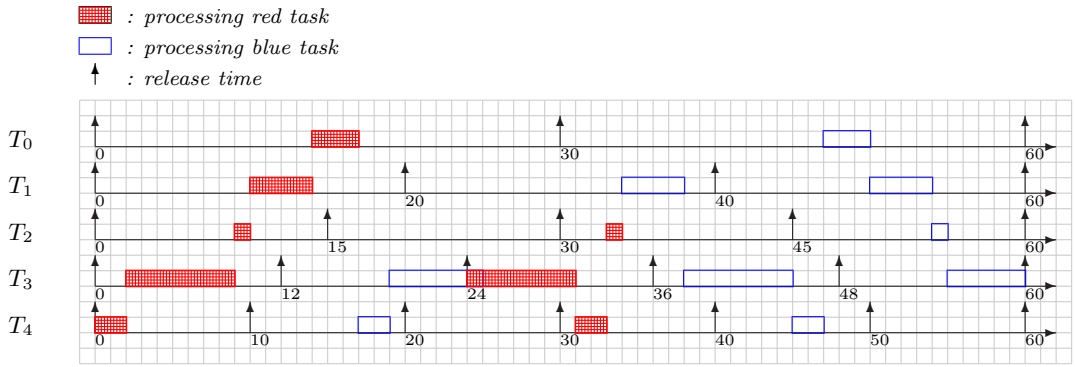

Figure 2: BWP scheduling algorithm ($s_i = 2$)

3.1.2 Blue When Possible (BWP) algorithm

The second algorithm studied is the Blue When Possible (BWP) algorithm which is an improvement of the first one. Indeed, BWP schedules blue instances whenever their execution does not prevent the red ones from completing within their deadlines. In that sense, it operates in a more flexible way. Deadline ties are still broken in favor of the task with the earliest release time. Figure 2 shows an illustrative example of BWP scheduling using the task set previously described in Table 1.

Compared with RTO, more task instances complete successfully with BWP. We observe that five violations of deadline relative to blue task instances occur at time instants $t = 24$ (task T_3), $t = 30$ (tasks T_2 and T_4) and $t = 60$ (tasks T_3 and T_4), thus reducing the QoS.

3.2 The EDL algorithm

The definition of the Earliest Deadline as Late as possible (EDL) algorithm makes use of some results presented by Chetto and Chetto in [5]. Under EDL, periodic tasks are scheduled as late as possible. An accurate characterization of the idle times during which the processor is not occupied is necessary. The authors introduced an *availability function* f_Y^X defined with respect to a task set Y and a scheduling algorithm X. $f_Y^X(t) = 1$ if the processor is idle at t, 0 otherwise.

So, for any instants t_1 and t_2, value of $\int_{t_1}^{t_2} f_Y^X(t)dt$ denoted by $\Omega_Y^X(t_1, t_2)$ gives the total idle time in $[t_1, t_2]$. f_Y^{EDL} can be described by means of the following two vectors:

- $\mathcal{K}$, called *static deadline vector*, represents the times at which idle times occur and is constructed from the distinct deadlines of periodic tasks.

- $\mathcal{D}$, called *static idle time vector*, represents the lengths of the idle times relating to time instants of vector $\mathcal{K}$.

The complexity of the EDL algorithm is $O(Kn)$ where n is the number of periodic tasks, and K is equal to $\lfloor \frac{R}{p} \rfloor$, where R is the longest deadline, and p is the shortest period [13]. We also recall the fundamental property relative to the optimality of EDL [5]:

Theorem 1. *Let X be any preemptive scheduling algorithm and $\mathcal{A}$ a set of independent aperiodic tasks. For any instant t,*

$$\Omega_{\mathcal{A}}^X(0, t) \leq \Omega_{\mathcal{A}}^{EDL}(0, t) \tag{1}$$

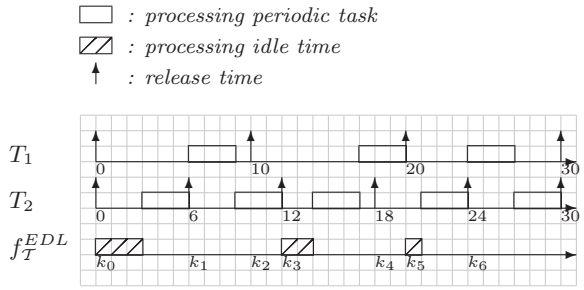

Figure 3: $f_{\mathcal{T}}^{EDL}$ computation produced at time zero

We give now an illustrative example of the computation of the idle times performed by EDL. Consider the periodic task set $\mathcal{T} = \{T_1, T_2\}$ consisting of two periodic tasks $T_1(3, 10)$ and $T_2(3, 6)$. The $f_{\mathcal{T}}^{EDL}$ computation produced at time zero is described in Figure 3.

The authors in [5] described how the EDL algorithm can be applied, first to the decision problem that arises when a sporadic time critical task occurs and requires to be run at an unpredictable time and secondly, to the scheduling problem that arises in a fault tolerant system using the Deadline Mechanism [**?**] for which each task implements primary and backup copies (the processor time reserved for the execution of the backup copies is realized with EDL and is reclaimed as soon as the primary task executes successfully).

In next sections, we are interested in using EDL first to simulate a schedule (RLP implementation) and then to derive a measure required for deciding whether a blue task can be accepted (RLP/T implementation).

4 The Proposed Algorithms

4.1 Red tasks as Late as Possible (RLP)

The main drawback of BWP relies on the fact that blue task instances are executed as background tasks. This leads to abort partially or almost completely executed blue task instances, thus wasting processor time.

4.1.1 Algorithm outline

The objective of RLP algorithm is to bring forward the execution of blue task instances so as to minimize the ratio of aborted blue instances, thus enhancing the actual QoS (*i.e.,* the total number of task completions) of periodic tasks. From this perspective, RLP scheduling algorithm, which is a dynamic scheduling algorithm, is specified by the following behaviour:

1. if there are no blue task instances in the system, red task instances are scheduled as soon as possible according to the EDF (Earliest Deadline First) algorithm.

2. if blue task instances are present in the system, these ones are scheduled as soon as possible according to the EDF algorithm (note that it could be according to any other heuristic), while red task instances are processed as late as possible according to the EDL algorithm.

Deadline ties are always broken in favor of the task with the earliest release time. The main idea of this approach is to take advantage of the slack of red periodic task instances. Determination of the latest start time for every red request of the periodic task set requires preliminary construction of the schedule by a variant of the EDL algorithm taking skips into

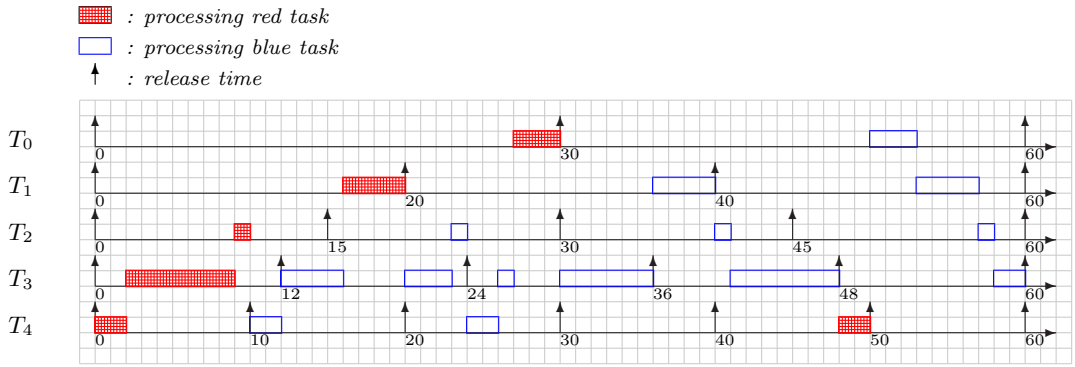

Figure 4: RLP scheduling algorithm ($s_i = 2$)

account [12]. In the EDL schedule established at time τ, we assume that the instance following immediately a blue instance which is part of the current periodic instance set at time τ, is red. Indeed, none of the blue task instances is guaranteed to complete within its deadline. Moreover, Silly-Chetto in [13] proved that the online computation of the slack time is required only at time instants corresponding to the arrival of a request while no other is already present on the machine. In our case, the EDL sequence is constructed not only when a blue task is released (and no other was already present) but also after a blue task completion if blue tasks remain in the system (the next task instance of the completed blue task has then to be considered as a blue one). Note that blue tasks are executed in the idle times computed by EDL and are of same importance beside red tasks (contrary to BWP which always assigns higher priority to red tasks).

4.1.2 Illustrative example

Consider once again the periodic task set $\mathcal{T}$ defined in Table 1. The relating RLP scheduling is illustrated in Figure 4. In this example, we can see that, thanks to RLP scheduling, the number of violations of deadline relative to blue task instances has been reduced to three. They occur at time instants $t = 40$ (task T_4), and $t = 60$ (tasks T_3 and T_4). Observe that T_3 first blue task instance which failed to complete within its deadline in the BWP case (see Figure 2), has enough time to succeed in the RLP case, since the execution of T_1 and T_0 first red task instances is postponed. Until time $t = 10$, red task instances are scheduled as soon as possible. From time $t = 10$ to the end of the hyperperiod (defined as the least common multiple of task periods), red task instances do execute as late as possible in the presence of blue task instances, thus enhancing the actual QoS of periodic tasks.

4.2 Red tasks as Late as Possible with blue acceptance test (RLP/T)

The main drawback of RLP relies on the fact that this algorithm attempts to execute blue task instances as soon as possible, at the risk of aborting them before their completion, thus generating a processor time wasting. This assessment led us to propose a novel algorithm named RLP/T (Red tasks as Late as Possible with blue acceptance Test).

4.2.1 Algorithm outline

Red tasks as Late as Possible with blue acceptance test (RLP/T) algorithm is designed to maximize the actual QoS of periodic task sets defined under skip constraints.

It acts as follows: red tasks enter straight the system at their arrival time whereas blue tasks integrate the system upon acceptance. Once they have been accepted, blue tasks are scheduled as soon as possible together with red tasks. Upon acceptance, blue tasks are again of same importance beside red tasks. Deadline ties are always broken in favor of the task with the earliest release time.

Whenever a new blue task enters the system, the idle times are computed using the EDL scheduler. In the EDL schedule established at time τ, we assume that the instance following immediately a blue instance which is part of the current periodic instance set at time τ, is also blue. Indeed, all blue task instances previously accepted at τ are guaranteed by the schedulability test they passed successfully. This one checks whether there are enough idle times to accommodate the new blue task within its deadline, as described in the following section.

4.2.2 Acceptance test of blue tasks under RLP/T

Now, we are ready to present the new feasibility test algorithm for the RLP scheduling scheme which, given any occurring blue task B is capable of answering the question "Can B be accepted ?". Notice that B will be accepted if and only if there exists a valid schedule, *i.e.*, a schedule in which B will execute by its deadline while red periodic tasks and blue tasks previously accepted, will still meet their deadlines. Let τ be the current time which coincides with the arrival of a blue task B. Upon arrival, task $B(r, c, d)$ is characterized by its release time r, its execution time c, and its deadline d, with $r + c \leq d$. We assume that the system supports several blue tasks at time τ. Each of them has been accepted before τ and has not completed its execution at time τ. Let denote by $\mathcal{B}(\tau) = \{B_i(c_i(\tau), d_i), i{=}1 \text{ to } blue(\tau)\}$ the blue task set supported by the machine at τ. Value $c_i(\tau)$ is called dynamic execution time and represents the remaining execution time of B_i at τ. A deadline occurs at d_i. We assume that $\mathcal{B}(\tau)$ is ordered such that $i < j$ implies $d_i \leq d_j$.

The acceptance test of blue tasks within a system involving RLP skippable tasks presented below in Theorem 2, is based on the one established by Silly-Chetto and al. [14] for the acceptance of sporadic requests occurring in a system consisting of basic periodic tasks (*i.e.*, without skips).

Theorem 2. *Task B is accepted if and only if, for every task $B_i \in \mathcal{B}(\tau) \cup \{B\}$ such that $d_i \geq d$, we have $\delta_i(\tau) \geq 0$, with $\delta_i(\tau)$ defined as:*

$$\delta_i(\tau) = \Omega^{EDL}_{\mathcal{T}(\tau)}(\tau, d_i) - \sum_{j=1}^{i} c_j(\tau) \tag{2}$$

$\delta_i(\tau)$ is called slack of task B_i at time τ which represents the maximum units of time during which the task could not be served by the processor without missing its deadline. $\Omega^{EDL}_{\mathcal{T}(\tau)}(\tau, d_i)$ denotes the total units of time that the processor is idle in the time interval $[\tau, d_i]$. The total computation time required by blue tasks within $[\tau, d_i]$ is given by $\sum_{j=1}^{i} c_j(\tau)$

The procedure that implements the acceptance test calls for the EDL algorithm for the computation of the total idle times which will be used to compute the slack of blue tasks. Then, this slack is compared to zero. Thus, the acceptance test proposed in this paper runs in $O(\lfloor \frac{R}{p} \rfloor n + blue(\tau))$ in the worst-case, where n is the number of periodic tasks, R is the longest deadline, p is the shortest period, and $blue(\tau)$ denotes the number of active blue tasks at time τ, whose deadline is greater or equal to the deadline of the occurring task. Note that this acceptance test could be implemented in $O(n + blue(\tau))$ by considering and maintaining to update additional data structures using slack tables, as proved in [15].

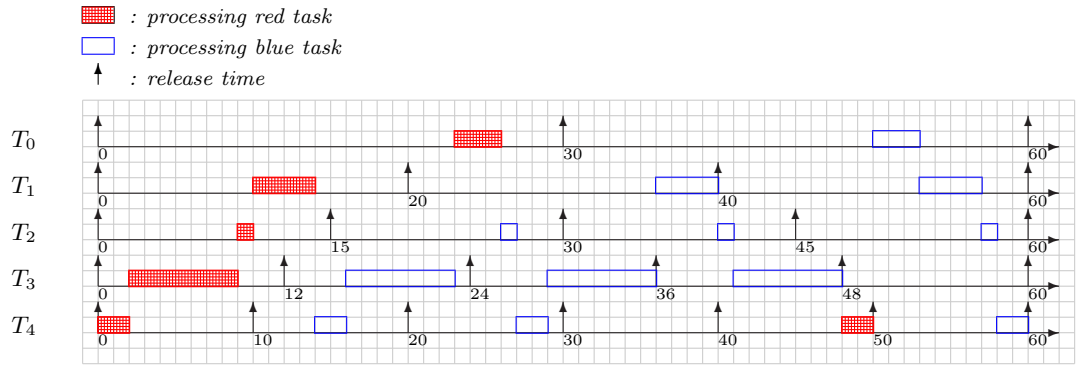

Figure 5: RLP/T scheduling algorithm ($s_i = 2$)

4.2.3 Illustrative example

RLP/T scheduling is illustrated in Figure 5 with the periodic task set $\mathcal{T}$ defined in Table 1. It is easy to see that RLP/T improves on both RLP and BWP. Only two violations of deadline relative to blue task instances are observed: at time instants $t = 40$ (task T_4) and $t = 60$ (task T_3). The acceptance test contributes to compensate for the time wasted in starting the execution of blue tasks which are not able to complete within their deadline. As we can observe, in the RLP case (see Figure 4), T_3 blue instance released at time $t = 48$ is aborted at time $t = 60$ (2 units of time were indeed wasted). Note that the rejection of this blue task instance, performed with RLP/T, contributes to save time used for the successful completion of T_4 blue instance released at time $t = 50$.

In section 6, we quantify more precisely the gain of performance of RLP/T upon RLP, BWP and RTO.

5 Applying theory to real-world problems

5.1 Multimedia applications

In order to understand the importance of CPU scheduling in multimedia real-time applications, it is useful to place the issue in context. Multimedia implies a certain amount of data to be handled within a specified time frame and requires a tremendous amount of resources to accommodate. For multimedia applications to function correctly, there must be a steady stream of data for the output devices to process. For the viewer to perceive continuous media such as movies or music, the output devices have to output new media within strict time constraints (e.g. 30 frames per second for video applications). Given this observation, one gets a better understanding of the crucial role of CPU scheduling in such applications.

Consider a simplified model of a real-time telesurveillance application, as represented in Figure 6. The relevant tasks are the *acquiring tasks* and the *display task*. Video data are first captured and digitized through video capture devices such as video cameras. Then, each video capture task "Acq_i" reads the input video buffer relative to its camera, thus periodically acquiring incoming frames. Downstream from the chain, another task named "*Display*" is in charge of continuously consuming frames from an output frame buffer and sending the acquired video frames to a final display device composed of various telesurveillance screens.

One important problem part of the management of telesurveillance systems is the data refreshing rate on the display device. Indeed, by definition, such a system must provide pictures as recent as possible to be useful. If there is no data for the output devices to process, there

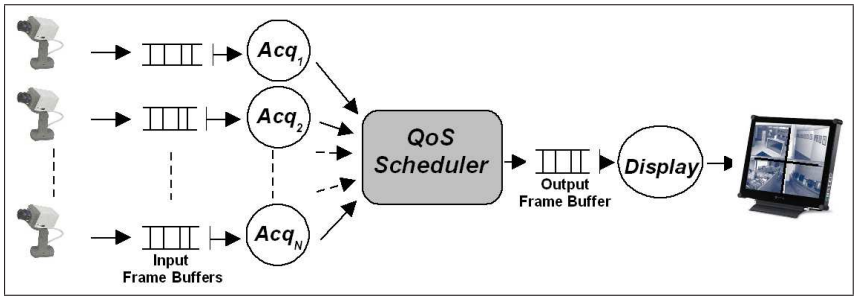

Figure 6: Simplified architecture of a real-time telesurveillance application

is buffer underflow. The media application will stall and wait for new data to be provided. Naturally, buffer underflow should be avoided whenever possible.

Scheduling implies multiplexing a resource among several tasks to ensure all throughput requirements are met. In the present case, the problem consists in ensuring an acceptable refreshing rate (*i.e.* a guaranteed QoS level).

5.2 Dynamic QoS scheduling

For multimedia applications, the CPU scheduler determines the resulting quality of service. The more CPU cycles allocated to a task, the more data can be produced, thus providing a better quality output. In the previous example, more CPU cycles produce more frames, thus allowing more frames per second to be displayed on the workstation monitor. However, if several videos are executing at the same time there may not be enough CPU cycles available to produce all the video frames requested. In this overloaded situation, the quality of service (*i.e.* frame rate) is reduced to a lower acceptable level, which results for instance to display an image at 15 frames per second instead of 30 frames per second. The resulting video just appears less smooth during the transient overload period.

Now, let us consider RLP or RLP/T algorithms for scheduling the application described in Figure 6. At initialization time, the application specifies a desired average rate of execution by appropriately setting the skip parameter s_i of each "Acq_i" task. RLP (or RLP/T) scheduler can be viewed as an EDF priority-based scheduler coupled with a skip-over rate regulator. That ensures every task not to be executed below a specified rate, whatever is the CPU workload. RLP and RLP/T superiority over RTO and BWP effectively results in a higher and guaranteed frame rate on the workstation monitors.

6 Simulation Study

In this section, we summarize the results of a simulation study which compares the performance of the different QoS scheduling algorithms. The objective is to maximize the actual QoS level of periodic tasks, i.e., the ratio of periodic tasks which complete before their deadline.

Experiments also evaluate the impact of the skip value for each algorithm, namely RTO, BWP, RLP and RLP/T.

6.1 Simulation context

The simulation context includes 50 periodic task sets, each consisting of 10 tasks with a least common multiple equal to 3360. Tasks are defined under QoS guarantees specified by uniform s_i. Their worst-case execution time is randomly generated and depends on the input setting

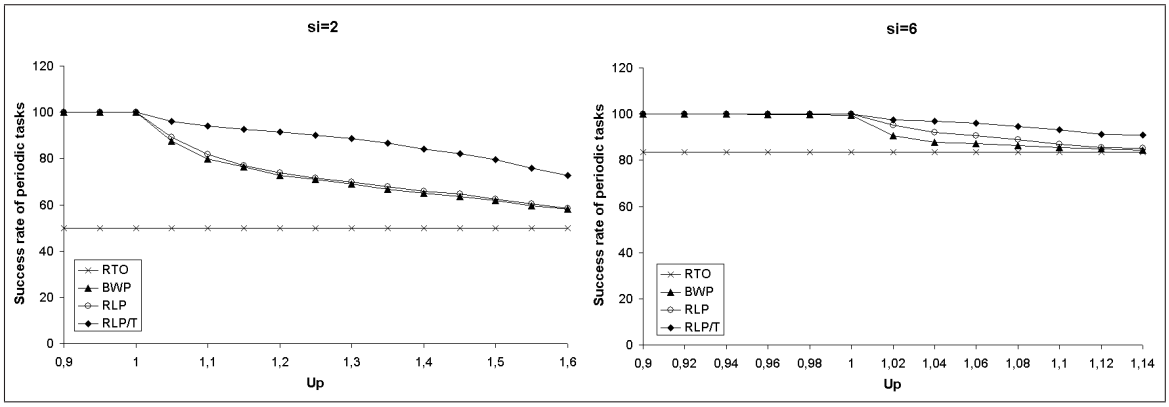

Figure 7: QoS of periodic tasks with low and high skip parameters ($s_i = 2$ and $s_i = 6$)

of the periodic load U_p. Deadlines are equal to the periods and greater than or equal to the computation times. Simulations have been processed over 10 hyperperiods.

6.2 Varying the periodic load

Measurements rely on the ratio of periodic tasks which complete before their deadline. The evaluation is done varying the periodic load U_p. The results obtained for $s_i = 2$ (one instance every two can be aborted) and $s_i = 6$ (one instance every six can be aborted) are described on Figure 7.

From the graphs, we can say that BWP, RLP and RLP/T outperform the RTO model in which the QoS level is still constant whatever is the periodic load applied. For $s_i = 6$, the actual QoS (which corresponds to the QoS guarantee) remains constant at a rate of 5/6=83%. The advantage of RLP over BWP is slight for low skip parameters, and more significant for high skip parameters. We note that the performance of BWP and RLP is dramatically worse than the one achieved by RLP/T. This result was expected because both BWP and RLP attempt to schedule blue instances that have not enough time for completing within their deadlines. This wasted time is not saved for executing other blue instances with closer deadline. In contrast, RLP/T finds a way of saving this CPU time by implementing an acceptance test for blue instances. We can observe that this gain of performance is all the more significant as the periodic load U_p is higher. For instance, in Figure 7, for $U_p \geq 120\%$, RLP/T enjoys more than factor $\frac{1}{4}$ success rate advantage over BWP. Moreover, we observe a very low gradient for the RLP/T curve which is not the case for other models. For $U_p = 150\%$ and $s_i = 2$, actual QoS levels for RTO and RLP/T are respectively equal to 50% and 84%, which figures the great predominance of RLP/T over RTO.

The variation of the skip parameter value shows that, for wide loads, the actual QoS of periodic tasks is all the more improved with RLP/T that the QoS constraint is smaller. For instance, for $U_p = 110\%$, RLP/T applied to periodic tasks with $s_i = 2$ will successfully process twice as many periodic instances over BWP, as with periodic tasks with $s_i = 6$. As we can see, the major difference in the performance between RLP/T and BWP appears not only for heavy loads but also for small value of s_i.

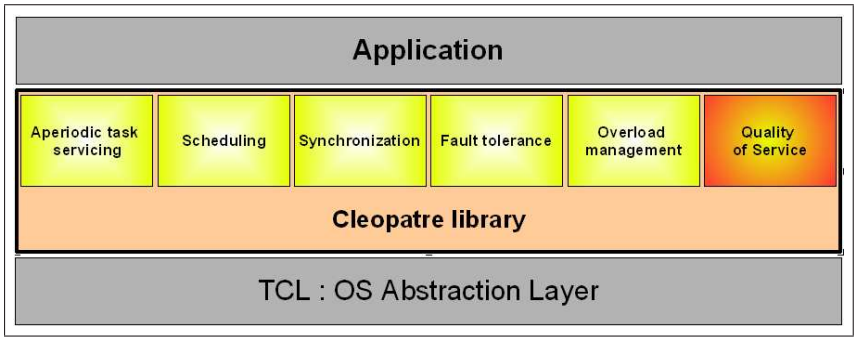

Figure 8: The CLEOPATRE framework

7 Integration into a Linux-based system

7.1 CLEOPATRE library

RTO, BWP, RLP and RLP/T algorithms have been integrated into a library of free software components called CLEOPATRE (Software Open Components on the Shelf for Embedded Real-Time Applications) [6]. This library, part of a French National Project [1], was designed to provide more efficient and better service to real-time applications. The purpose was to enrich the real-time facilities of real-time Linux versions, such as RTLinux [18] or RTAI [9]. RTAI was the solution adopted for this project because we wanted the CLEOPATRE components to be distributed under the LGPL[2] license which is also the one used in the RTAI project.

The CLEOPATRE library whose framework is shown in Figure 8 offers selectable COTS (Commercial-Off-The-Shelf) components dedicated to dynamic scheduling, aperiodic task service, resource control access and fault-tolerance. Components are totally independent from the kernel and the hardware. Reusability of the components with another hardware and OS is made possible by just adapting the OS abstraction layer in the TCL component. This component hides the specific features of each platform, so that the run-time components can be implemented in a portable fashion and adapted to the target's processor architecture and board.

RTO, BWP, RLP and RLP/T can be found in a new shelf called "Quality of Service"'. Final users can then build their own customized applications through the flexible and easy-to-use interface provided by the CLEOPATRE framework.

8 Conclusions

This paper pointed out the need of more flexible scheduling solutions for real-time applications dealing with multimedia and active monitoring systems. Our main contribution was actually to propose and validate new scheduling algorithms, namely RLP and RLP/T. Their purpose is to enhance the QoS of periodic tasks that allow skips (i.e the ratio of task instances that do execute within their deadline) while providing a QoS guarantee (i.e the ratio of task instances that must complete within their deadline). We considered a real-world problem (*i.e.* a multimedia application) to bring to light how these algorithms can be implemented in practice in order to provide a better QoS. Simulation results show that the improvements with both RLP and RLP/T are quite significant compared with basic algorithms. These new QoS functionalities are available under Linux/RTAI. Our future work includes extending these QoS scheduling algorithms to multiprocessor systems.

[1] work supported by the French research office, grant number 01 K 0742

[2] Lesser General Public License

References

[1] G. Bernat, A. Burns, Combining (n/m)-hard feadlines and dual priority schaduling, *18th IEEE Real-Time Systems Symposium*, pp 46-57, 1997.

[2] G. Bernat, A. Burns, A. Llamosi, Weakly-hard real-time systems, *In IEEE Transactions on Computers*, Vol. 50, No. 4, pp 308-321, 2001.

[3] G.-C. Buttazzo, M. Caccamo, Minimizing Aperiodic Response Times in a Firm Real-Time Environment, *IEEE Trans. Software Eng.*, Vol. 25, No. 1, pp 22-32, 1999.

[4] M. Caccamo, G.-C. Buttazzo, Exploiting skips in periodic tasks for enhancing aperiodic responsivess, *18th IEEE Real-Time Systems Symposium*, 1997.

[5] H. Chetto, M. Chetto, Some Results of the Earliest Deadline Scheduling Algorithm. *In Proceedings of the IEEE Transactions on Software Engineering*, Vol. 15, No. 10, pp 1261-1269, 1989.

[6] T. Garcia, A. Marchand, M. Silly-Chetto, Cleopatre: a R&D project for providing new real-time functionalities to Linux/RTAI. *5th Real-Time Linux Workshop*, 2003.

[7] M. Hamdaoui, P. Ramanathan, A Dynamic Priority Assignment Technique for Streams with (m,k)-firm deadlines. *IEEE Transactions on Computers*, Vol. 44, No. 4, pp 1443-1451, 1995.

[8] G. Koren, D. Shasha, Skip-Over Algorithms and Complexity for Overloaded Systems that Allow Skips. *16th IEEE Real-Time Systems Symposium (RTSS'95)*, Pisa, Italy, 1995.

[9] P. Mantegazza, E. Bianchi, L. Dozio, M. Angelo, D. Beal, DIAPM. RTAI Programming Guide 1.0, *Lineo Inc.*, 2000.

[10] A. Marchand, M. Silly-Chetto, QoS Scheduling Components based on Firm Real-Time Requirements, *ACS/IEEE International Conference on Computer Systems and Applications (AICCSA'05)*, Le Caire (Egypt), 2005.

[11] A. Marchand and M. Silly-Chetto, RLP: Enhanced QoS Support for Real-Time Applications, *11th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'05)*, Hong-Kong, 2005.

[12] A. Marchand, M. Silly-Chetto, Dynamic Real-Time Scheduling of Firm Periodic Tasks with Hard and Soft Aperiodic Tasks. *Journal of Real-Time Systems*, Vol. 32, No. 1-2, pp 21-47, 2006.

[13] M. Silly-Chetto, The EDL Server for Scheduling Periodic and Soft Aperiodic Tasks with Resource Constraints, *Journal of Real-Time Systems*, Vol. 17, pp 1-25, 1999.

[14] M. Silly-Chetto, H. Chetto, N. Elyounsi, An Optimal Algorithm for Guranteeing Sporadic Tasks in Hard Real-Time Systems. *IEEE Symposium on Parallel and Distributed Processing*, pp 578-585, 1990.

[15] T. Tia, J. Liu, J. Sun, R. Ha, A Linear-Time Optimal Acceptance Test for Scheduling of Hard Real-Time Tasks, *Technical report, Department of Computer Science*, University of Illinois at Urbana-Champaign, IL, 1994.

[16] R. West, C. Poellabauer, Analysis of a Window-constrained scheduler for real-time and best-effort packet streams, *21st IEEE Real-Time Systems Symposium*, Orlando, USA, 2000.

[17] R. West, Y. Zhang, K. Schwan, C. Poellabauer, Dynamic window-constrained scheduling of real-time streams in media servers, *IEEE Trans. on Computers*, Vol. 53, pp. 744-759, 2004.

[18] V. Yodaiken, The RTLinux Approach to Real-Time, *FSMLabs Inc.*, 2004.

[19] Y. Zhang, R. West, X. Qi, Avirtual deadline scheduler for window-constrained service guarantees, *Tech. Rep. 2004-013*, Boston University, 2004.

Audrey Marchand
University of Nantes
Laboratoire d'Informatique de Nantes Atlantique
2, rue de la Houssinière - BP 92208
44322 Nantes Cedex 03,
FRANCE
E-mail: audrey.marchand@univ-nantes.fr

Maryline Chetto
University of Nantes
Institut de Recherche en Communications et Cybernétique de Nantes
1, rue de la Noe
44321 Nantes Cedex 03
FRANCE
E-mail: maryline.chetto@univ-nantes.fr

Received: May 22, 2008

Audrey Marchand graduated in Computer Engineering at the Ecole polytechnique of the University of Nantes (France), in 2002. After getting a Master Degree in Applied Computer Science at Ecole Centrale de Nantes in 2003, she received the PhD degree in October 2006 from the University of Nantes. From October 2006 to August 2007, she hold a Post-Doc researcher position at the Polytechnic University of Valencia, Spain. She is currently an Associate Professor at the University of Nantes, France. Her research interests include real-time scheduling theory, OS service mechanisms, quality of service guarantees in real-time systems, and Linux-based real-time OSes and applications.

Maryline Chetto received the degree of Docteur de 3ème cycle in control engineering and the degree of Habilitée Diriger des Recherches in Computer Science from the University of Nantes, France, in 1984 and 1993, respectively. From 1984 to 1985, she held the position of Assistant professor of Computer Science at the University of Rennes, while her research was with the Institut de Recherche en Informatique et Systèmes Aléatoires, Rennes. In 1986, she returned to Nantes and is currently a professor with the Institute of Technology of the University of Nantes. She is conducting her research at IRCCyN. Her main research interests include scheduling and fault-tolerance technologies for real-time applications. She has published more than 60 journal articles and conference papers in the area of real-time operating systems. She is the leader of a French national R&D project, namely Cleopatre, supported by the French government, which aims to provide free open source real-time solutions.